Intrinsic Compilation Model to enhance Performance of real time application in embedded multi core system

Sumalatha Aradhya [#1], Dr.N K Srinath [*2]

[#] F Research Scholar, Department of Computer Science and Engineering, R V College of Engineering, Mysore Road, Bengaluru, Karnataka, India
[1] sumalatha.aradhya@cedlabs.in
[*] Professor and Dean Academics, R V College of Engineering, Mysore Road, Bengaluru, Karnataka, India
[2] srinathnk@rvce.edu.in

***Abstract*— The embedded multi core system has critical performance issues. This is due to extra optimization codes added by the compiler. In many cases, performance becomes challenging due to increased cyclomatic complexity of the embedded software especially in the release version of the code. To ease the extra complexities of the software, a better optimization approach through intrinsic compilation model proposed in the paper. The proposed intrinsic compilation model handles software to hardware integrity complexities through vector dynamic interface algorithm discussed in the paper. The vector dynamic interface algorithm resolves the conflicts of optimization across multi core target by using optimizer conflict resolver algorithm. The optimizer conflict resolver algorithm computed by the recurrence logic derived through probabilities using vector rule algorithm. The intrinsic compilation model makes use of effective partitioning logics to obtain optimized vector code data. The bench mark shows improved speed up results between static code and vector code.**

Keyword - compiler, intrinsic programming, parallel processing, performance, optimization, simulation

I. INTRODUCTION

The compiler set up with high optimization levels is not used in embedded integrated environment due to enormous usage of volatile data present in the code. Volatile provides the safe way to use the data to process interrupts and inter process communication. It is required to utilize the available resources up to optimum extent as well as to obtain parallel performance invading Amdahl's law [1]. While handling interrupts and inter process communication data, it is required to face the issues occurred due to memory constraints, scheduler delays, in efficient task distribution across processors and improper load balancing across multi cores. These issues induce the performance problems. As a solution, an intrinsic compilation model is designed and implemented to address the performance issue and this model is discussed in the paper. The three layered architecture representation of the intrinsic compilation model depicted in Figure 1.

The embedded software faces performance issues due to higher degree of cyclomatic complexity [2]. The reasons for the raised complexities are due to unaddressed deactivated codes, dead codes and spaghetti codes used in the embedded software. Though most of the time compiler successfully eliminates dead or spaghetti codes, the whole program optimization adds extra compiler codes in the software to optimize the code. The extra code added by the compiler induces run time execution problems resulting in insidious bugs and anomalies in the execution environment. Hence, an embedded industry today strives for optimized code developers as most of performance fixes can be taken care of while at coding level [3]. The software developers need training or extra skills to adapt code optimization techniques and these extraneous skills set kills lot of human resource effort hours and effort values indeed practically leading to wastage of resource time and effort values. Thus, there was need of code hinting tool to aid the performance of code and the programmer. In perspective of solution to obtain optimal code, vector dynamic interface is implemented. The vector dynamic interface uses static object code library generated by native compiler and static analysis data obtained through absInt [4]. The intrinsic compilation subsystem components and their interfaces depicted in Figure 2

An enormous amount of work been done in optimizing the compiler to achieve speed up. To analyse and improve the performance, the existing approach is auto vectorization. VTune Performance Analyzer [5], Intel Thread Building Blocks [6], Terra [7] etc., have set the trend mark in optimizing the code using auto vectorization techniques [9]. However, due to resource constraint and critical system requirements, most of the time auto vectorization tools are not exercised in real time systems. To analyse the spaghetti codes, dead codes etc., an existing approach is to set the native compiler option with code elimination switches. In real time systems, especially in safety critical systems a certified tool chain such as KPIT's GNU compilers, Reneases Rx compilers etc. are widely used.

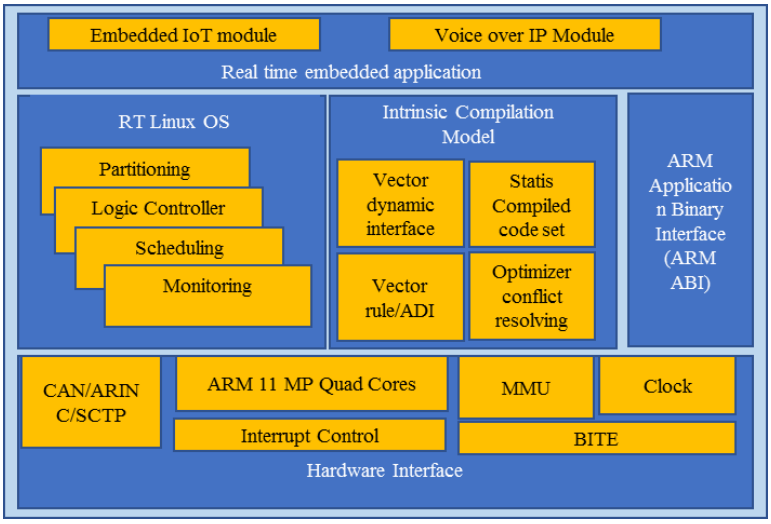

Fig. 1. 3-layer architecture model of intrinsic compilation model

However in practice, a loop level optimization resulted in consumption of more execution time especially when executing while loop. The existing approach to deal with loop optimization are to use loop unrolling optimization techniques and this optimization results vary from one target to another target and are inconsistent. For loop level dependencies, a partition implementation strategy proposed by Samuel Larsen et.al [10] through selective vectorization. But, prototyping compiler transformations though easier at standalone applications, require high performance computations with fine grained control to access memory [13, 18]. Hence, it is required to hint the programmer to optimize the case while coding itself to reduce the release time hurdles. One way to hint programmer is to use the modelling based approaches. For modelling, a real time constraint logic through RT parallel computations suggested by Peter Hui et.al [15]. However, the worst case analysis needed calculation methods such as program flow analysis and low level analysis. Falk et.al provided the compiler framework to calibrate worst case execution time for real time system in their paper [21] and an extra time was considered for unnecessary measurements.

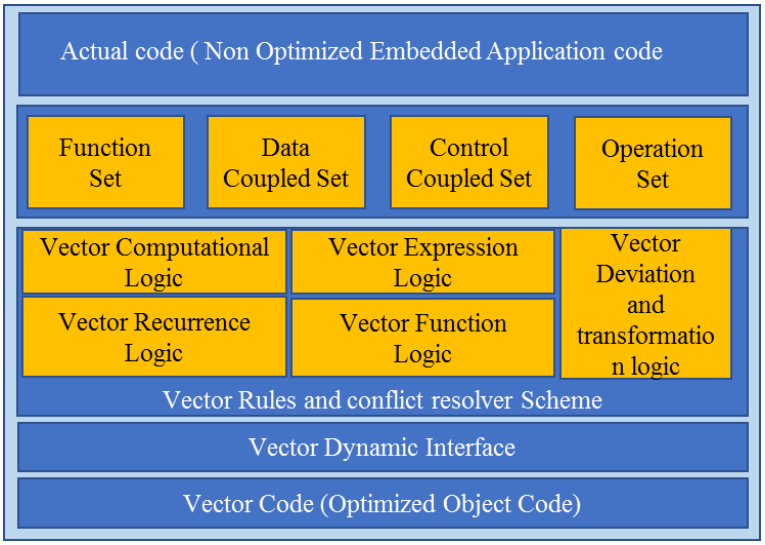

Fig. 2. Subsystem components and interface of intrinsic compilation model

II. Related Work

The optimal alignment of data across memory models with conflict free access been proposed by A. Seznec et al [16] has implementation logic specific to vectors on cache, but it has limited scope in real time systems. The traditional embedded compilers do not provide features of designated initializations and providing automation through scripting or manual effort is timid task. We have partially adopted the extending g++ methodology used in the paper [7, 8] for vector code. Sherwood et al. [12] proposed dynamic optimization approaches to aid multi-threaded interference in parallel programs with code segmentation logics. To achieve the best feasibility and performance maths specific applications, hybrid Intel TBB with MPI for parallelization approach [3] and Intel Cilk Plus Array Notation for vectorization using ivdep pragma directives been used [22].

But, Intel TBB library sometime will not be compatible with non-Intel specific targets especially real time embedded systems. TLOG provides a code generator for parameterized loops where loop sizes are symbolic parameters [13] and has limitations with vector arrays. The Table I depicts the comparisons of parallel software interface comparisons in brief [21, 22].

TABLE I. Parallel software Interface comparisons with Hardware abstraction

Sl No	Software Interface	Data Parallelism	A-synchronized task parallelism	Host and/or device	Abstraction of memory hierarchy	Explicit data mapping and movement
1	Cilkplus	Cilk_for, array operations, elemental functions	Cilk_spawn/sync	Host only	Data	N/A (host only)
2	CUDA	<<…….>>	Async kernel launching and memcpy	Device only	Blocks, threads, shared memory	cudaMemcpy function
3	C++ 11	X	std::thread,std::async, std::future	host only	Data	N/A(host only)
4	OpenACC	Kernel/parallel	Async/wait, acc parallel, acc data	Host only	#pragma acc	N/A(Host only)
5	OpenCL	Data and task based parallelism at kernel level	Command specific apis,clEnqueueNDKernelRange	Host only	__global, __local, __kernel,__constant,__private	N/A(Host only)
6	OpenMP	Data parallelism	Fork-join model	Host and device	#pragma directives	SIMD program
7	Pthreads(POSIX threads)	Data	syscall interface, wrapper library functions	Host and device	_pthread	pthread_t, system call routines
8	Thread Building Block	Task parallelism	Templates	Host	Data	Vector program
9	Unified Parallel C	Data	Wrapper function with bupc_	Host	Data	Implicit
10	Charm++	Data parallelism	Fork-Join mode, Adaptive Message passing interface, method invoking procedures	Host only	Data	N/A(host only)
11	Coarray Fortran	Data parallelism	sync all, barrier like SPMD constructs	Host only	locks, point to point event synchronization using events, cofence, finish	Implicit
12	OpenHMPP(Hybrid Multicore parallel Programming)	Data parallelism	Pure function, no static , volatile variables, codelet RPC remote execution <LabelOfGroup> callsite, synchronize, region	Host only	#pragma hmpp	allocate, release, advancedload, delegatedstore
13	PVM(Parallel	Task	Library routines	Host &	PVM_	Explicit

Sl No	Software Interface	Data Parallelism	A-synchronized task parallelism	Host and/or device	Abstraction of memory hierarchy	Explicit data mapping and movement
	Virtual Machine)	parallelism		Device		
14	RaftLib	Task and pipeline parallelism	raft::kernel	Host	Compute graph of kernel	Explicit
15	ZPL(Zebra Programming Language)	Data parallelism	Command specific codes through ^ symbol, device.cpcl_	Host and device	Command specific	Explicit
16	Chapel	Data, task and nested parallelism	Data driven on clauses, SPMD like parallelism : Coforall loc in Locales do	Host and device	Data	Implicit
17	X10	Data parallelism	APGAS (Asynchronous Partitioned Global Address Space) and using async S statement	Host	Interfaces e.g., interface Normed {...}	Implicit

We found that other important points needed to achieve better performance were the management of the data inside cache memories, affinity of threads into the cores of the systems and scheduling of the threads runtime execution [4]. Furthermore, specifically for our application it was also important to assure the reproducibility of results across different parallel executions and across executions with different number of threads. Among the three different implementations based, respectively on OpenMP [24], Intel Cilk Plus [22], and Intel TBB [6], the last one has the best results in terms of performance with the minimum number of changes required to C++ application source code. When comparing the performance on systems with different number of sockets and high number of cores, we found that the hybrid MPI implementation improved the performance when combined with Intel Cilk Plus and Intel TBB parallel implementations [9, 24,25]. The application scaled very close to the theoretical expectation, also when using SMT, reaching a speed-up of about 35x on 32 SMT enabled cores [25].

Our intrinsic compilation module acts as a plug in to the existing software and enhances the code optimization by hinting the programmer with intrinsic code sets. A mature vectorization algorithm by name vector dynamic interface algorithm proposed in this paper helps compiler transformations effectively to speed up performance of high end real time embedded applications.

III. INTRINSIC COMPILATION MODEL DERIVATION

Given any real time based embedded application, static or actual code of application gets fragmented into partition set. To obtain the partition data set from the repository of data obtained through static analyser AbsInt which is based on abstract interpretation [4], the best possible partitioning set through approximation methods on variant set is used. Section A-F provides detailed derivation of the approximation logic used to obtain optimal partitioned data set.

A. Derivation of Vector Function and Computation Logic

Hypothetically in congruence, an application as super set has F function blocks, D data paths, Op operational logics and control logics and the group is shown as super set, $SetA_{partition}$, in equation (1).

$$SetA_{partition} = \{F, \{D, Op\}\} \cdots\cdots\cdots\cdots\cdots\cdots (1)$$

For each vector sequence, function path F in the super set $SetA_{partition}$ is derived as shown in eqs. 2 and 3[26, 28],

$$f_\infty(x) = 1 - \exp\wedge\left(-2x \wedge 2\right) \cdots\cdots\cdots\cdots\cdots\cdots (2)$$

$$where, x \geq 0$$

For fixed series $f_n(x)$, function derived as proportional to $1/\sqrt{(n)}$ for all random variable x and probability of median of such occurrences represented as, $Fn \leq t/\sqrt{n}$. The statistics of derived function estimated as, $\sqrt{n}$ (j/n - f (x_t)) where, $1 \leq t \leq n$. For each sequence of function derived, the set of computations for fixed bound derived as [26],

$$F = (f(x2) + \lambda 2) * (f(x1) + \lambda 1) * (f(x0) + \lambda 0) * n. \cdots\cdots\cdots (3)$$

Finding the functions to be optimized for $SetA_{partition}$ as set F is possible, with propagation of sequences derived from the probability of occurrences as per equations (2) and (3). F is a nonlinear set of data and each occurrence f(x) is an active function. The occurrence of duplication is filtered by the iterative mapping of functions stacked. When the function set has identical mapping, the propagation from one function to another function directly in both forward and backward direction. Both the functional units result in maximum error when identical mapping is found. The individual cases are discussed as below:

Case i. Fixed bound computation set, Fis 0, implies a plain linear set with no identical mapping. The linear dependency of the function blocks are shown in Figure 3. It shows the case where the functions are mapped linearly and the value of zero is added to the set F, that is, f(x) is evaluated to 0 as per equation. (3).

Case ii. F or f(x) > 1, implies the backward value is increased and the occurrence of identical mapping with more conflicts. The error rate increases linearly and leads to exploding of functionality errors. The Figure. 4 depict the identical mapping found for the functions mapped in the sequence.

With respect to the function mapping found for identical tasks, the different probability of occurrence of activities and computations in the function set is represented in Figure 5.

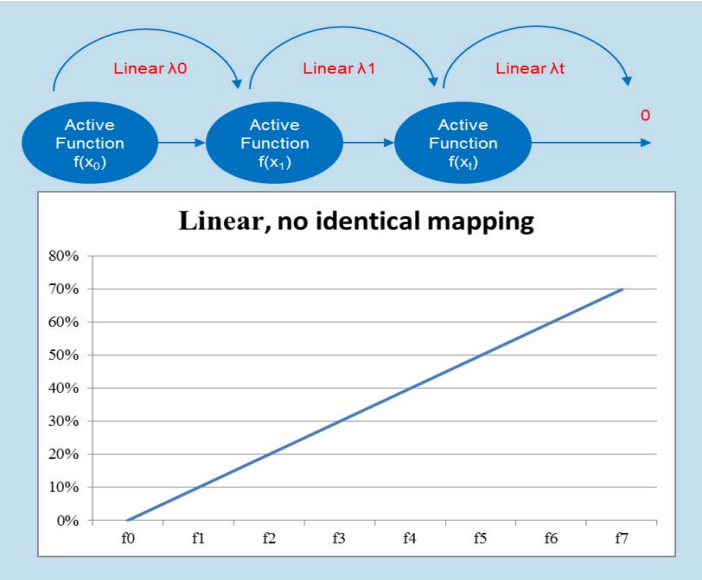

Fig. 3. Subsystem components and interface of intrinsic compilation model

The actual flow path of the computation is shown in Figure. 5 and is predicated with the possible occurrences such as repetition before and after ʎ change, pre active computation with single repetition and full pre active computation set. Once such data set is obtained, the iterative mapping and collision between function sets are detected further by using vector rule algorithm and optimal conflict resolver algorithm discussed in section III.B and III.C.

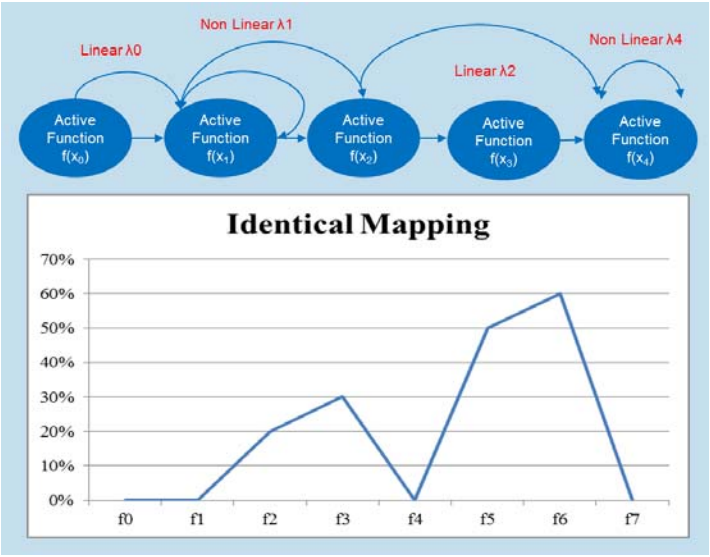

Fig. 4. Subsystem components and interface of intrinsic compilation model

B. Vector Rules Algorithm

To obtain sequence of distinct optimized function elements, i.e., Os, following vector rule algorithm applied.

Step 1: For a sequence of (x1, x2…xt), function f(x1, x2…xt) computed such that Optimal set of function f (o1, o2…ot) is obtained. Thus, f(x1, x2…xt) = f(o1, o2…ot), provided xi and oi have same relative ordering. For all recurrence r, function f exist such that f! =r.

Step 2: Initialize recurrence r with t and set function set f to 0.

Step 3: Os is the maximum set i.e., f (o1, o2…os), set to maximum set Os

Step 4: Swap static sequence with optimized sequence

Step 5: Decrease recurrence r by one until r reaches 1, otherwise return to step 2.

All remaining partition set elements follow the vector rule algorithm correspondingly. The probability of possible optimizer conflicts obtained as container array conArray. The collisions occurred during static to optimal code transformation obtained as per logic referring to recurrence logic derived at section C-F. A container array is represented as conArray[p], and p represents probability occurrences. Initially, p is set with value 1 and then, for all p with (pi, pi-1… p0), the probability conArray set obtained as, (conArray (pTransitiontj + (collision1 + 1/Transitiont) - (p/Transitiont))*conArray[p-1]), where, Transitiont represents transition sequences and collision1 represents first collision occurred. If array data, conArray[p], obtained as 10-20, then reset conArray[p] to 0. Otherwise, compute optimizer conflict algorithm.

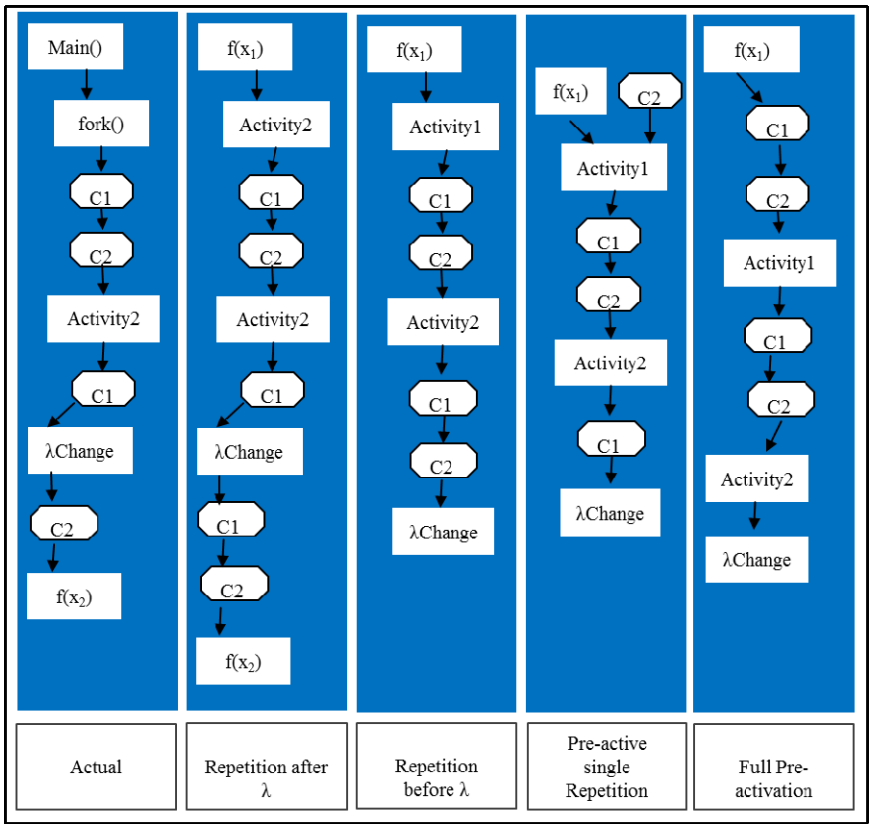

Fig. 5. Probability of pre active computation occurrences in function

C. Optimal Conflict Resolver Algorithm

Following the vector rule described in section III.B and III.C, the container array obtained i.e., conArray is used for resolving conflict probabilities by using optimal conflict resolver algorithm. The optimal result of collision free set is stored in variable Optim_Collision_Result. The probability p is used through iteration variable Transition_iteration_t and the optimal result is obtained until maximum transition count, i.e., Transition_tmax is reached. The optimal conflict resolver algorithm is given as below.

Step 1: Initialize Optim_Collision_Result <- 0 as Transition_t <- 1 and probability p <- p + 1

Step 2: Assign p <- p + 1

Step 3: Optim_Collision_Result = Optim_Collision_Result + conArray[p]

Step 4: if Optim_Collision_Result > Transition_iteration_t go to step 5, 6 otherwise goto step 2

Step 5: Optim_Collision_Result <- (n - p - 1) & (1 - Optim_Collision_Result)

Step 6: Increment Tranisition_iteration_t by 1

Step 7: Repeat Step 2 to 4 until Transition_t <- Transition_tmax <= Transition_iteration_t

The vector rule algorithm and optimal conflict resolver algorithm mentioned in section III.B and III.c are used in vector dynamic interface algorithm depicted in section III.D. The vector dynamic interface algorithm makes use of mathematical model derived in section III.

D. Vector Dynamic Interface Algorithm

The vector dynamic interface model design is depicted in Figure 6. The vector dynamic interface is implemented through the optimizer conflict resolver scheme. The vector computation imposes vector schemes for all one to one mapped prepared/Unprepared data set retrieved through input fragmentation framework [8], with the set of expressions and sub expressions, user defined functions, control logics, operations, interrupts, pragmas and macros etc. The permutation logic for sequence of distinct elements of computation with relative ordered vector set exercised through vector dynamic interface algorithm

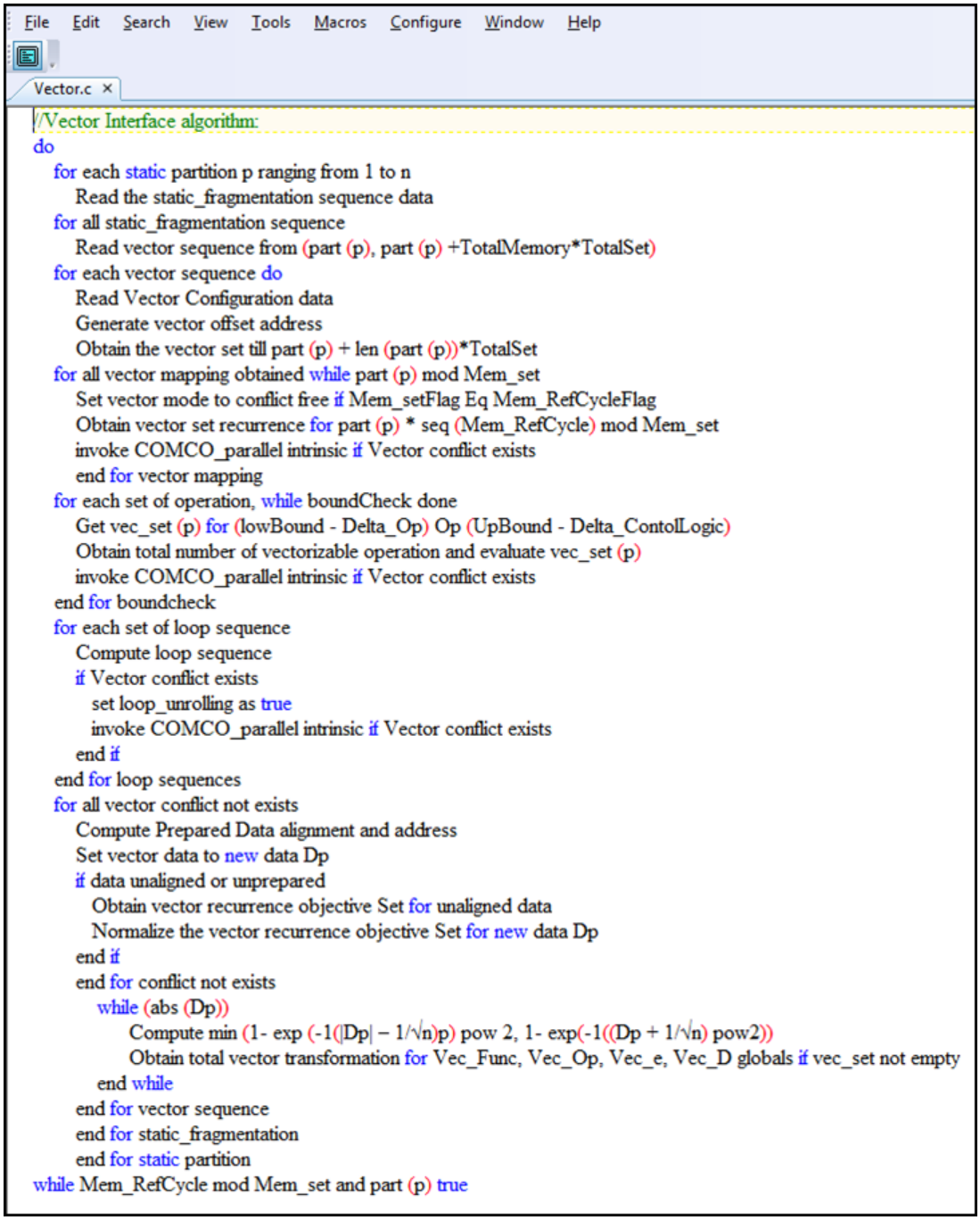

Fig. 6. Vector dynamic interface algorithm

The results and discussions of execution of implementation logic depicted in section II.H

E. Derivation of the total vector deviation and transformation logic

The maximum deviation, maxDev, calculated when vectorized set F >static set F and minimum deviation, minDev, when vectorized set F <static set F as shown in eqns. 4 and 5 respectively [26].

$$maxDev = \sqrt{n}\,(F_n(x) - F(x)) \quad \text{.....(4)}$$

$$minDev = \sqrt{n}\,(F(x) - F_n(x)) \quad \text{.....(5)}$$

F. Derivation of recurrence logic

The static loop sequence i till r recurrences and with address sequence m_r is framed as,

Step 1: loop i: 1 to N

Step2: loop j: 1tor

Step3: $m_r = m_r\,(i - 1)\,r + j$

With the vectorization, the static loop sequence reduced to half with offset addresses m_{r1} and m_{r2} is generated as,

Step1: loop i: 1 to N/2

Step2: loop j: 1 to r

Step3: $m_{r1} = m_{r1}\,(i - 1)\,r + j$

Step4: $m_{r2} = m_{r2}\,((2(i - 1)\,r + rn) / 2 + j)$

Such vectorization transformation is provided as a hint to optimize the code to the programmer while coding. To obtain the vector optimized code, the derivation logic given in section III.E and III.F are applied. From the actual static set to the new vector optimized set, the total deviation is calculated as per Eqns. 4 and 5 given in section III.E. The computation activity with recurrence sequence is vectorized by using transformation logic shown in III.F. By using all the derivation logic on the set of embedded application is tested as explained in section G with experimental set up as shown in Figure 7.

G. Experimental Setup , Results and Discussion

The result, vector optimized code, obtained using a vector dynamic interface algorithm and intrinsic interface logic in synonym with vector rule explained in section III. The mathematical model formulated in section III is used in the implementation of intrinsic compiler optimization model for obtaining the result that is to obtain vector optimized code. The test environment set up is shown in Figure 7. The RTSim and Matlab simulator provides the graphical interface. For the test purpose, we have considered an Embedded IoT module. The test executed through visual basic test script file containing real time test scenarios. The test result obtained shown in Figure 10

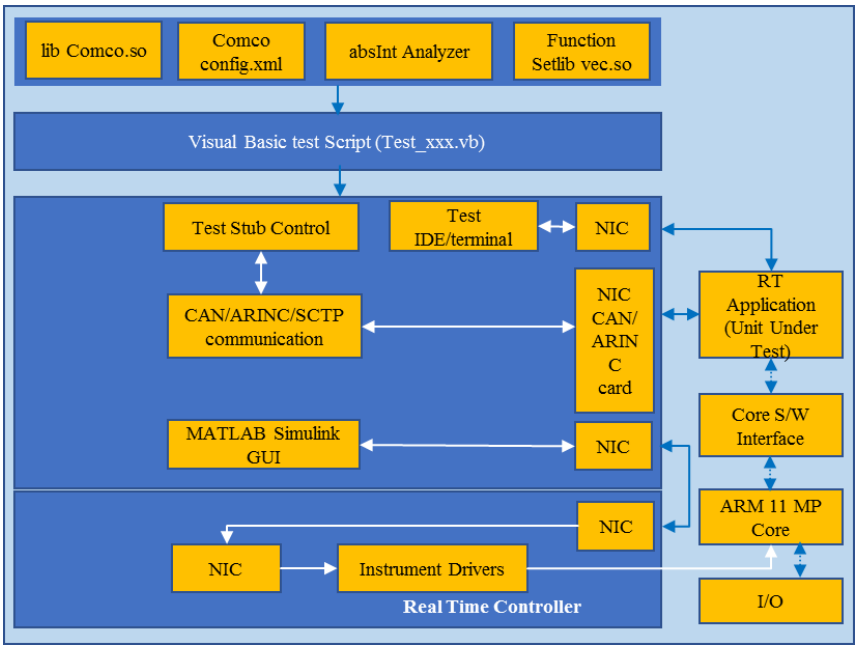

Fig. 7. Test Environment Set Up

The analysis of the vector computation set obtained using the mathematical model derived in section III is represented in Figs. 8 and 9. The analysis shows performance achieved through set partition and memory usage by both static code and vector optimized code. It is been proved that intrinsic compilation model when used across the static partitioned real time embedded software reduced the compiler consternation such as speed up, memory constraint and throughput across ARM MP Core processor. Speed up of static code set is compared with vector code and result margins shows that performance of vector set well than that of static code set. The representation of vector result as shown in Figure 10 demonstrates that optimal speed up is achieved. The result data contains the partitioned data set spaces obtained for functions, expressions, control logics and computation. The analysis chart shown in Figure 8 represents the partition data set obtained from static analysis of code run. The analysis chart shown in Figure 9 represents the partition data set obtained after implementing the intrinsic compilation model framework. It is clear that the after the optimization the partition data set occupied less memory space than that of actual code partition data set. The updated code set is plugged in through library libVec.so and libComco.so [8]. As shown in Figure 7, the test procedure is written using visual basic script and the test is driven through the compiled set of libraries and by test stub control triggered through communication links such as CAN [29] and ARINC [30] in safety real time applications. The result obtained at the GUI front end simulated through MATLAB shows the memory space occupied by each partition data set namely Function Set, Operation Set, Expression Set and Control Set

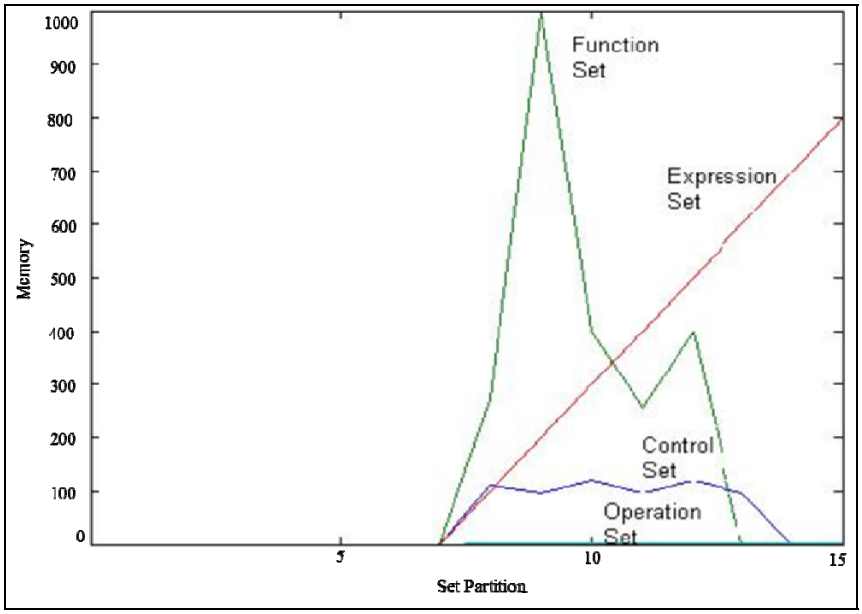

Fig. 8. Partition Setup – Static or Actual code

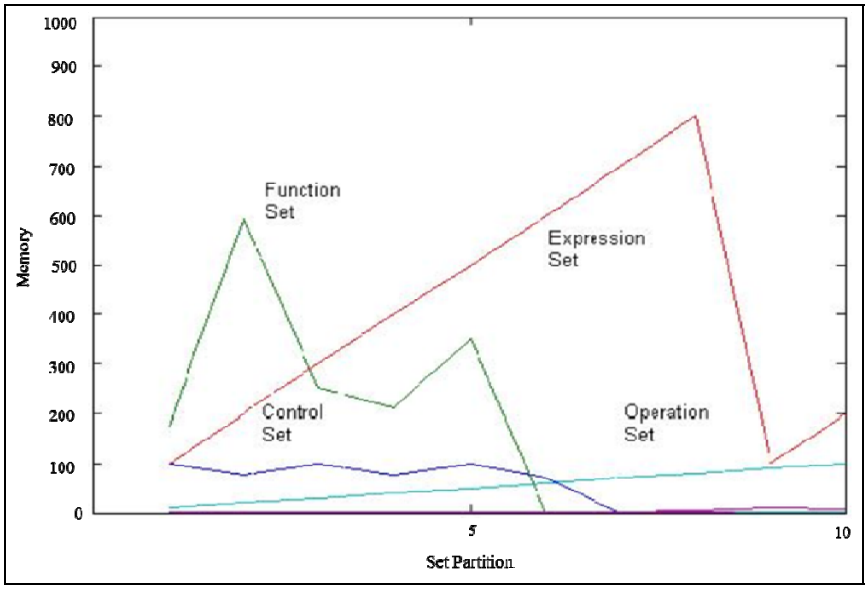

Fig. 9. Partition Setup – Optimized or vector code

The partition data set computed through the derivation logic explained in section II and the execution time of the actual and optimized comparison data with respect to partition data set is shown in Figure 10. From Figure 10, it is clear that vector or optimized partition data set statistics are better than static or actual partition data set. For the experiment, the real time application from integrated control module software is considered. Through the static analyser, AbsInt[4] , we obtain the static partition code data set and the re run of the code attempted after the static analysis and implementation of intrinsic compilation model logics at the test IDE.

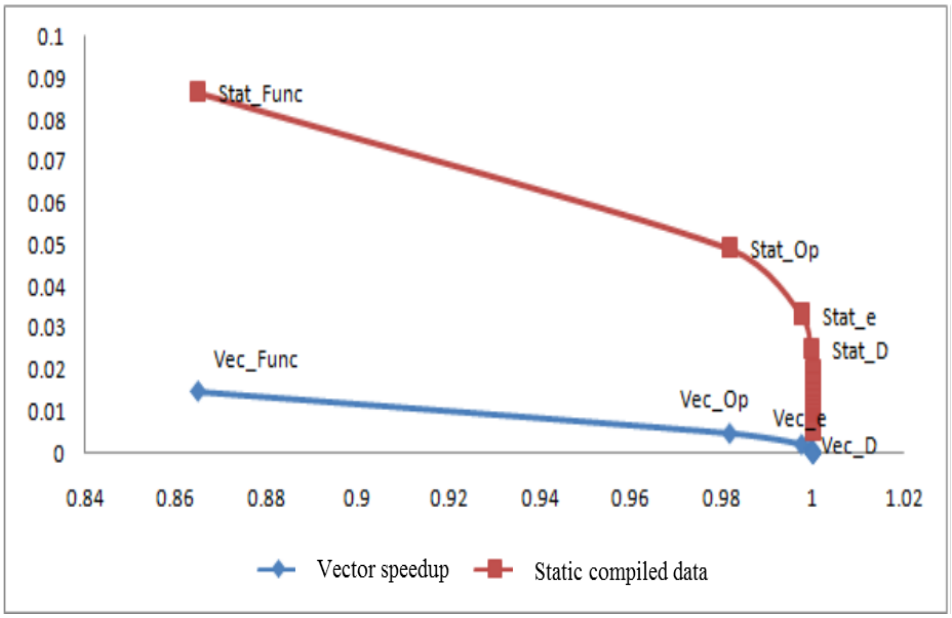

Fig. 10. Speed up Comparison of static code Vs. Vector code

IV. CONCLUSION

To obtain the better proximity in compiler optimization, proposed intrinsic compilation model is a remarkable approach for resolving the code complexities, memory constraints and scale up performance consternations. The result proves that execution of optimal code has provided better speed up than normal static code. The adaption of intrinsic compilation model helps any embedded coder to write optimized code. The adaption of the vectorization logics through an interactive feedback to improve the loop recurrence optimization while coding is proposed in the paper. The compilation model avoids unnecessary code measurement time of compiler and thus providing proliferation to improve the speed up of real time application in use.

ACKNOWLEDGMENT

The project idea is dwelled and nurtured while working for real time embedded project at CED LABS, Tumkur Karnataka. We would like to thank all the technical guidance, motivation and support provided by the development and testing team.

REFERENCES

[1] Mark D. Hill and Michael R. Marty, "Amdahl's Law in the Multicore Era", Computer, 41 (7), 33-38, 2008, [Online]. Available: http://research.cs.wisc.edu/multifacet/papers/tr1593_amdahl_multicore.pdf
[2] Accounting for Secondary Uncertainty: Efficient Computation of Portfolio Risk Measures on Multi and Many Core Architectures by Blesson Varghese and Andrew RauChaplin, WHPCF 13 Denver, Oct 2013, ACM, 1310.2274
[3] Multi-dimensional sla-based resource allocation for cloud computing systems by H. Goudarzi and M. Pedram, Proceedings first Intl workshop on data center performance (DCPerf11), held in conjunction with ICDCS2011
[4] Kastner.D. and Ferinand, C, Efficient verification of non-functional safety properties by Abstract interpretation: Timing, Stack Consumption, and Absence of Runtime Errors, Proceedings of the 29th International System Safety Conference ISSC2011, Las Vegas.
[5] Intel (R) VTune (TM) Performance Analyzer 9.0 by Intel Corporation, IntelWebSite. [Online]. Available:http://intel-r-vtune-tmperformance-analyzer.software.informer.com/9.0/
[6] James Reinders, Intel Thread Building Blocks, Technical Report, Oreilly publications, 2007
[7] Zachary DeVito, James Hegarty, Alex Aiken, Pat Hanrahan, and Jan Vitek Terra: A Multi-Stage Language for High-Performance Computing Stanford University Purdue University PLDI13 June 16-22, 2013, ACM 978-1-4503-2014-6/13/0.
[8] Sumalatha Aradhya, and Dr. Srinath N K, Proliferation framework on input data set to improve memory latency in multicores for optimization , Proceedingsof IACC , June 2015,IEEE publications, 978-1-4799-8047-5, DOI: 10.1109/IADCC.2015.7154712.
[9] A.J.C. Bik, the Software Vectorization Handbook. Applying Multimedia Extensions for Maximum Performance, Technical Report, Intel Press, June, 2004.
[10] Samuel Larsen, Rodric Rabbah and Saman Amara singhe, Exploiting Vector Parallelism in Software Pipelined Loops ,Technical Report, MITCSAIL-TR-2005-039, June 2005, MIT Computer Science and Artificial Intelligence Laboratory
[11] Hemang Mehta, S J Balaji, and Dharanipragada Janakiram, Extending Programming Language to Support Object Orientation in Legacy Systems, Department of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai - 600036.

[12] T. Sherwood, E. Perelman, G. Hamerly, and B.Calder. Automatically characterizing large scale program behavior. In Proceedings of the 10th international conference on Architectural support for programming languages and operating systems, pages 4557, ACM publications, 978-0-7695-4428-1/11, 2011 IEEE, DOI 10.1109/ISPA.2011
[13] Sverre Jarp, Alfio Lazzaro, Andrzej Nowak, Liviu Valsan ,Comparison of Software Technologies for Vectorization and parallelization , White-paper as part of the collaboration between CERN openlab and Intel SSG, CERN openlab, September 2012 version 1.0
[14] D. Kim, L. Renganarayanan, D. Rostron, S.Rajopadhye, and M. M. Strout. Multi-level tiling: M for the price of one. In SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing, pages 1-12, New York, NY, USA, 2007. ACM.
[15] Peter Hui, Satish Chikkagoudar, C. Artho and P.C. Olveczky ,A Formal Model for Real-Time Parallel Computation (Eds.): EPTCS 105, 2012, pp. 3955,doi:10.4204/EPTCS.105.4.
[16] A. Seznec and R. Espasa, Conflict free accesses to strided vectors on a banked cache, IEEE Trans. On Comp., vol. 54, pp. 913 - 916, 2005.
[17] S. J. Pennycook, C. J. Hughes, M. Smelyanskiy and S. A ,Exploring SIMD for Molecular Dynamics Using Intel R Xeon Processors and Intel R Xeon PhiTMCoprocessors , Parallel Computing Lab, Intel Corporation, IEEE publication, 2013
[18] Y. Zhang, S. Misra, D. Honbo, A. Agrawal, W.keng Liao, and A. N. Choudhary, Efficient pairwise statistical significance estimation for local sequence alignment using GPU, in ICCABS, 2011, pp.226231
[19] Rafiqul Zaman Khan ,Current Trends in Parallel Computing, International Journal of Computer Applications (0975 8887) Volume 59 No.2, December 2012
[20] R. M. Rabbah and K. V. Palem. Data for design Space optimization of embedded memory systems. ACM Transactions on Embedded Computing Systems, (2):132, May 2003
[21] Falk, Heiko, and Paul Lokuciejewski. "A compiler framework for the reduction of worst-case execution times.", Technical Report, Real-Time Systems 46.2 (2010)
[22] I. Karlin, J. McGraw, J. Keasler, B. Still, uning the LULESH Mini-app for Current and Future Hardware, LLNL-CONF-610032, NECDC 2012, Livermore, CA, United States
[23] HPCWebsite. [Online]. Available: https://www.hpcwire.com/2015/03/02/a-comparison-of-heterogeneous-and-manycore-programming-models
[24] François Broquedis, François Diakhaté, Samuel Thibault, Olivier Aumage, Raymond Namyst, and Pierre-André Wacrenier R. Eigenmann and B.R. de Supinski ,Scheduling Dynamic OpenMP Applications over Multicore Architectures (Eds.): IWOMP 2008, LNCS 5004, pp. 170–180, 2008. @ Springer
[25] Jarp, Sverre & Lazzaro, Alfio & Leduc, Julien & Nowak, Andrzej. (2011). Evaluating the scalability of HEP software and multi-core hardware. Journal of Physics: Conference Series. 331. 10.1088/1742-6596/331/5/052009.
[26] Richard G. Brown, " Advanced Mathematics- precalculus with Discrete Mathematics and Data Analysis", 1st ed. ,McDougal Littell Inc., A Houghton Mifflin Company, Illinios. ISBN:0-395-77114-5
[27] C L Liu and D P Mohapatra, "Elements of Discrete Mathematics-a computer oriented approach", 3rd ed., Tata McGraw-Hill Publishing company Limited, New Delhi, India, ISBN-13:978-0-07-066913-0.
[28] Kenneth Hoffman and Ray Kunze, "Linear Algebra", 2nd ed., PHI Learning Private Limited, NewDelhi, 2009, ISBN: 978-81-203-0270-9.
[29] An ISO Standard, ISO 11898-1:2015, Road Vehicles- Controller Area Network (CAN)-Part 1: Data link layer and physical signaling", The ISO website. [Online]. Available:www.iso.org/standard/63648.html
[30] ARINC Specification 429, Part 1-17. Annapolis, Maryland: Aeronautical Radio, Inc. 2004-05-17. pp. 78–116

AUTHOR PROFILE

Mrs. Sumalatha Aradhya received B E degree from Dr. Ambedkar Institute of Technology, Bangalore in year 2000 and M Tech from M V J College of Engineering, Bangalore in year 2006 and currently perceives her PhD in the field of parallel computing from R V College of Engineering, Bangalore. The author has 16+ years of experience with diversified exposures to telecom, avionics and memory computing areas across the industries such as IntelliNet Technologies, Bangalore; L & T Infotech, Bangalore; HCL Technologies, Bangalore and Quest Global, Bangalore. Her research interests are compilers, high performance computing, and embedded systems

Dr. N K Srinath is currently working as Dean of Academics at R V College of Engineering, Bangalore. He has 33 plus year of experience in teaching and his area of research are systems engineering and operations research. He has more than 52 international journal publications and he is author of several text books related to microprocessors and data base systems. He served as advisory committee member for various national and international proceedings, and was part of expert committee member of UGC as chairman and is active member of several education bodies' advisory committees.